FMFP SUMMARY '16

SIMONE GUGGIARI
ETH ZÜRICH 2016
LEGI # 14-932-867

1. INTRODUCTION

Correctness Proofs
Algorithm
turing machine, register machine, Java program, function, recursion } ≠ lvl of abstraction
- compute value
- update mem

Imperative vs Functional program

- change state
- depends on state

- recursive
- what to compute, not how
- no side effects
- function/values

Type system → referential transparency (same value in every context)

History

- lambda calculus
- LISP
- FP/ML
- → Haskell GHC/ghci

Expression Evaluation

EAGER (call by value) LAZY (call by need)

SYNTAX & TYPES

2D LAYOUT → indentation ␣ \t

more indentation opens block, less closes

TYPES → must match

Int $\{-2^{29}, \dots, 2^{29}-1\}$

operators
+ - * ^ div mod abs
→ infix 3+5 | 7 `mod` 2
→ prefix (+) 3 5 | mod 7 2
have different binding strength
> >= < <= == /= (unequal)
(return bool)
&& || not binds stronger
== ⟺ iff

Bool True, False

Char 'a', 'b', ..., 'A', 'B', ..., '\t', '\n'.

String "hello", "123", "a"

Double 0.34 -2.85e03

TUPLE (String, Int, Int) ("Veli", 1234, 5678)
at least 2 elem → ("foo") is not

TYPE CONSTRUCTOR
T_i is type → $(T_1, \dots, T_n)$ is tuple type
$v_i :: T_i$
↳ is type of → $(v_1, \dots, v_n) :: (T_1, \dots, T_n)$

nestable (3, ("Hi", True)) :: (Int, (String, Bool))

Functions → can take/return tuples

patterns } can use both → variables/const/data constructors
guards } for function def → bool expr

```
fun :: M1 M2 ... Mn
  lg1  = e1
  lg2  = e2
  :
  lgM  = eM
  otherwise = e   -- optional
```

Scope
- **global** – can be called from anywhere
  ```
  f x y
  g x = ... h x
  ```
- **local**
  ```
  let x1 = e1
      :
      xn = en
  in e
  ```

Input Output

() unit type in Haskell

do ← return

IO tag means it has side effects
↳ remains tagged
(can compute only in IO blocks)

2. NATURAL DEDUCTION

soundness, completeness

Formal proof $\mathcal{L} = \{\oplus, \otimes, +, \times\}$ • deductive rules

if / then

α: if +, then ⊗
β: + × (+ ⊗ and ×, then ⊕)
γ: if ⊗ and ×, then ⊕
δ: + holds

$\frac{+}{\otimes}\alpha$ $\frac{\otimes \quad \times}{\oplus}\gamma$ $\frac{+}{\times}\beta$ $\frac{}{+}\delta$ (axiom)

⚠ δ: assume + only when proving ⊕

⊢ "follows" (derivation under assumption)

PROPOSITIONAL LOGIC

prop built from collection of var, with • conj • disj • impl ... → p8 def

$\mathcal{V}$
$\mathcal{L}_P$ $\mathcal{V}, \bot, \wedge, \vee, \to$

valuation $\sigma : \mathcal{V} \to \{\text{True}, \text{false}\}$

satisfiability ⊨ ⊆ Valuations × $\mathcal{L}_P$
↳ "is a model of"

e.g. $\sigma \models X$ iff $\sigma(X) = \text{true}$
$\sigma \not\models \bot$ $\forall \sigma$

formula
- satisfiable
- tautology

Semantic entailment $A_1 \dots A_n \models A$ ⇔ $\sigma \models A_1, \dots, \sigma \models A_n$ then $\sigma \models A$ (truth tables)

Syntactic entailment ⊢ (derivation rules)

decidability

sequent: assertion $A_1, \dots, A_n \vdash A$

Axiom: starting point

Proof: derivation tree, root A

$\frac{}{A, \dots \vdash A}\,Ax$

if all rules preserve semantic entailment, then proof is sound

→ right associative
∧ binds stronger than ∨, →
¬ is the strongest

Intuitionistic – Classical logic adds $\frac{}{\Gamma \vdash A \vee \neg A}\,TND$

$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\,RAA$

Safe rule is safe if we only enlarge Γ or can get conclusion back

FIRST ORDER LOGIC (FOL)

terms $x \in V$, $f^n(x_1, \dots x_n)$, x_i : term, $f^n \in \mathcal{F}$

formulae $\bot$, $P^n(x_1, \dots, x_n)$ — , $P^n \in \mathcal{P}$, $A \circ B$ | $\circ = \vee \wedge \rightarrow$

, $Q\, x.A$ | $Q = \forall \exists$

$\mathcal{F}$ f^i | const are 0-ary

$\mathcal{P}$ P^i

bound / free

α-conversion = rename (only bound)

Quantifiers extend to the right as possible

SEMANTICS

structure interpretation $S = \langle U_S, I_S \rangle$ ↳ universe ↳ mapping

$I_S(P^n)$ relation

$I_S(f^n)$ function

$I = \langle S, v \rangle$

$v : \mathcal{V} \rightarrow U_S$ is valuation

value under interpretation

$I(t)$

interpretation can be a **model** $\langle S, v \rangle \models A$

change var $v[x \rightarrow a]$, valuates v' such that $v'(x) = a$

$\models A$ if every interpretation is valid

dense order → between any 2 elem there is another

× sparse

Substitution $A[t/x] = A[x \leftarrow t]$ (substitute, apply)

⚠ **capture** (don't instantiate free var with same name as bound → α-conv)

EQUALITY

extend FOL with '$=$'

rules: ref sym trans

structural, weak ind

4. LISTS & ABSTRACTION

List: T is type → [T] is type

$[] :: [T]$

$x :: T,\ xs :: [T] \rightarrow (x : xs) :: [T]$

functions: must specify • how to compute with explicit • other case (interpret)

(++) $[T] ++ [T]$

(:) $T : [T]$

pattern matching →

Patterns

• Constants	-2, 'a', True	succeeds if equal
• Variables	x, foo	succeeds, with binding
• "Wild card"	_	_, no binding
• Tuples	$(P_1, P_2, \dots, P_k)$	succeeds if all match
• Lists	$(P_1 : P_2)$	succeeds if non-empty and all match

⚠ **no multiple occurrences / type mismatch**

• can't build using functions $(x ++ y, z)$

• no duplicates $[x, y, x, z]$

List comprehension

CORRECTNESS - INDUCTION

for Lists: • prove $P([])$

• prove $P(xs) \rightarrow P(x : xs)$

difficulties: • select predicate variable

3. CORRECTNESS

- termination
- functional behavior

TERMINATION

if f is defined in terms of functions g_i and each terminate, so does f

⚠ recursion → args smaller along well founded order (no ∞ decreasing chain)

$>$ well founded $\Rightarrow >^+$ well founded

Types of reasoning • equation reasoning • cases $(P \vee \neg P)$

Induction $\forall n \in \mathbb{N}.\ P(n)$ | • base case: $P(0)$ • step case: $P(n) \rightarrow P(n+1)$

Noetherian: assumption is that $P(m)$ holds $\forall\ m < n$

$\forall n\ (\forall m < n\ P(m) \rightarrow P(n))$